Documentation for SimCommand.h and SimCommand.c

Steven Andrews, © 2004

See the document "LibDoc" for general information about this and other libraries.

```
#include "queue.h"
#include "string2.h"

typedef struct cmdstruct {
  float on;
  float off;
  float dt;
  int oni;
  int offi;
  int dti;
  int invoke;
  char *str;
  char *erstr; } *cmdptr;

typedef struct cmdsuperstruct {
  queue cmd;
  queue cmdi;
  int (*cmdfn)(void*,cmdptr,char*);
  void *cmdfnarg;
  int iter;
  int nfile;
  char root[STRCHAR];
  char froot[STRCHAR];
  int *fsuffix;
  char **fname;
  FILE **fptr; } *cmdssptr;

cmdptr scmdalloc(void);
void scmdfree(cmdptr cmd);
cmdssptr scmdssalloc(int (*cmdfn)(void*,cmdptr,char*),void *cmdfnarg,char
    *root);
void scmdssfree(cmdssptr cmds);
int scmdqalloc(cmdssptr cmds,int n);
int scmdqalloci(cmdssptr cmds,int n);
int scmdstr2cmd(cmdssptr cmds,char *line2,float tmin,float tmax,float dt);
void scmdpop(cmdssptr cmds,float t);
int scmdexecute(cmdssptr cmds,float time,int donow);
int scmdsetfroot(cmdssptr cmds,char *root);
int scmdsetfnames(cmdssptr cmds,char *str);
int scmdsetfsuffix(cmdssptr cmds,char *fname,int i);
int scmdopenfiles(cmdssptr cmds,int vb);
FILE *scmdoverwrite(cmdssptr cmds,char *line2);
FILE *scmdincfile(cmdssptr cmds,char *line2);
FILE *scmdgetfptr(cmdssptr cmds,char *line2);
void scmdoutput(cmdssptr cmds);
```

Requires: <stdio.h>, <stdlib.h>, <string.h>, "VoidComp.h", "SimCommand.h", "Zn.h", "queue.h", "string2.h".
Example program: Smoldyn

History: Routines moved to this library from *Smoldyn* 1/10/04. Moderate testing. Added invoke member to command structure 1/20/04; also changed declaration of command executing function. Made more changes to the command superstructure, added some functions, and modified others 1/22/04. Changed scmdstr2cmd 6/24/04 so that it now allocates the command queue or expands the queue as needed. Also added integer queue stuff to commands and command superstructure and erstr to commands.

When writing simulation programs, it has proven useful to include a runtime command interpreter in the program so that various commands can be executed at specific times during the simulation. These commands are stored as strings and are passed on to a parser and executer at the proper time. This library contains most of the framework necessary for this interpreter. As a primary use of commands is to output simulation results to text files, the command framework also manages a list of output files.

Data structures

The structure cmdstruct, pointed to with cmdptr, contains the infomation for one command. on is the time that the command turns on, off is the time that it turns off, dt is the time step between command executions, invoke is the number of times that the command has been invoked so far (it equals one for the first command call), and str is the command string. iter can be used instead of dt to indicate that a command should be run every iter iterations. Command execution intervals are never shorter than dt but are sometimes longer than dt because they can only be executed at the times when cmdcheck is called. Note that the cmdstruct owns the string, meaning that the string is allocated when a cmdptr is allocated and freed when the cmdptr is freed. erstr is storage space for an error message that can be passed from the command back to the calling program.

cmdsuperstruct, which is pointed to by cmdssptr, is a structure that contains the list of runtime commands, the address of a function that is supposed to execute them, and information about the output files. cmd is the regular queue of commands, sorted in order of their next exection times. cmdi is the queue of commands that are run every iter iterations and for which dt is ignored. In the queues, the object key is the on value of the command and the object item is a pointer to the command structure. cmdfn is a pointer to the function in the main program that is called to take care of commands. It is sent the argument cmdfnarg, which is unchanged by the routines here, the command, and the command string; see below. nfile is the number of output files, root is a root name used before froot, which is another root name and is used for all output files, fname is a list of file names for the various output files, fsuffix is a list of file name suffixes, and fptr is a list of file streams for the output files. Complete file names are a concatenation of root, froot, the file name, and the suffix if there is one. Usually, root is the directory in which the configuration file is located, and froot is a subdirectory for output results. The

command superstructure owns all lists and memory pertaining to output files, but cmdfn and cmdfnarg are merely pointers that are neither allocated nor freed in this library.

The function in the main program that takes care of commands is called docommand in *Smoldyn*. It separates the command string into the first word, which says what the command is, and the rest of the string which contain the parameters for the command, and then it calls the appropriate function to take care of the command. docommand is made available to the SimCommand library by sending its address to scmdssalloc as &docommand during initial structure setup. It is called later on, as needed, by scmdexecute. In this calling, docommand is sent a void* type conversion of sim, which is a structure for the entire simulation parameters and state, a pointer to the command that is to be executed (cmd), and the command string. In this case, the command string is always equal to cmd->str, and so is redundant. However, some commands can invoke other commands directly, in which case they call docommand with a valid string but either the original command or a NULL value for the cmd parameter. This means that all commands need to be able to handle cmd being NULL or the command string in cmd being different from the string in line. For example, the conditional command in *Smoldyn* called "ifno" first checks the condition and then, if appropriate, it calls docommand with the remainder of the command string.

Functions

```
cmdptr scmdalloc(void);
```
scmdalloc allocates a command structure, including the string and the error string. The strings are allocated to the fixed size STRCHAR, which is defined in the file string2.h to be 256.

```
void scmdfree(cmdptr cmd);
```
scmdfree frees a command structure.

```
cmdssptr scmdssalloc(int (*cmdfn)(void*,cmdptr,char*),void *cmdfnarg,char
*root);
```
scmdssalloc allocates a minimal command superstructure. cmdfn should be sent in pointing to a function that can execute the commands and cmdfnarg is the first argument of that function. For example, in the *Smoldyn* program, the cmdfn is sent in as &docommand and cmdfnarg is sent in as (void*)sim, because sim is a structure that contains all information about the current state of the simulation and is required for most commands. root is the file directory root. The only memory that is allocated is for the superstructure itself. In particular, the command queues are not allocated.

```
void scmdssfree(cmdssptr cmds);
```
scmdfree frees a command superstructure and all internal elements except for cmdfn and cmdfnarg.

```
int scmdqalloc(cmdssptr cmds,int n);
```

scmdqalloc allocates the command queue to size n and sets up the queue indexing parameters. It returns 0 for no error, 1 for insufficient memory, and 2 for no cmds. This function is called automatically by scmdstr2cmd, so there is no longer any need for it to be called from externally.

`int scmdqalloci(cmdssptr cmds,int n);`

scmdqalloci allocates the command queue cmdi to size n and sets up the queue indexing parameters. It returns 0 for no error, 1 for insufficient memory, and 2 for no cmds. This function is called automatically by scmdstr2cmd, so there is no need to call it from externally.

`int scmdstr2cmd(cmdssptr cmds,char *line2,float tmin,float tmax,float dt);`

scmdstr2cmd takes in a string in line2, parses it for a command type, timing, and command string, creates a new command for it, and adds it to the proper command queue. The queue is automatically created or expanded if needed. For the command timing of the floating point types (b, a, @, and i), this routine also needs to know the simulation start, stop, and time step parameters given in tmin, tmax, and dt. The format of line2 needs to have one of the following forms:

cmd b *string*	executes once before tmin (at tmin-dt)
cmd a *string*	executes once after tmax (at tmax+dt)
cmd e *string*	executes every time step
cmd @ *time string*	executes once at time given
cmd n *int string*	executes every *n*'th time step
cmd i *on off dt string*	executes at *on*, and every *dt* until *off*

The function can return any of several error codes: 0 is no error, 1 is memory allocation failed, 2 is cmds was set to NULL, 3 is error in line2 format, 4 is command string is missing from line2, 5 is command time step was set ≤0, 6 is the command timing type character was not one of those recognized, and 7 is a failure to insert the command in the command queue because memory could not be allocated for either a new queue or a larger queue. A change as of 6/24/04 is that the timing types e and n are now exact because they use integer arithmetic rather than floating point arithmetic; this is most useful for unequal length time steps.

`void scmdpop(cmdssptr cmds,float t);`

scmdpop removes all commands from the regular queue that are for time t or before, without executing them. The routine can be used after the simulation to avoid executing simulation time commands after an early exit from the simulation loop. It does not do anything to commands in the integer queue.

`int scmdexecute(cmdssptr cmds,float time,int donow);`

scmdexecute removes and executes all commands from the command queues that have times that are less than or equal to time for the floating point queue, or iteration counters less than or equal to the current iteration number for the integer queue. Commands that should be repeated in the future are put back in the proper queue with the execution time or iteration updated to the previous requested value

plus the command time step and with the invoke member incremented. The return value codes are essentially the same as those that are returned from the command executing function given in cmdfn. They are 0 for continue, 1 for a non-fatal error occured with at least command that was attempted, 2 for simulation should terminate, or 3 for stop this time step but continue the simulation (used for pausing). If the return value is 1, an error message is sent to stderr that says which command failed as well as what the error string contains if it was used. This function sends all commands to the command function listed in cmdfn. donow is a flag that produces normal operation, described above, when it equals 0; when it is 1, all remaining commands in the queue are executed immediately and are not put back in the queue.

```
int scmdsetfroot(cmdssptr cmds,char *root);
```

scmdsetfroot sets the file root element of the command superstructure to the string that is sent in. If it had already been set before with this command, the function returns 1 to indicate an error and otherwise it returns 0. Also, it returns 1 if cmds is NULL (and any previous string is written over).

```
int scmdsetfnames(cmdssptr cmds,char *str);
```

scmdsetfnames inputs a list of file names in str as words separated by spaces (if a file name has a space in it, this routine won't recognize the name correctly). It counts the number of names in the list, sets the nfile element of the command superstructure, allocates the fname and fptr lists, and copies the names to fname. The files are not opened. The routine returns 0 for success, 1 for inability to allocate memory, 2 for a file name that could not be read, 3 if this function has been called before (in which case the previous entries are unchanged), or 4 if cmds is NULL.

```
int scmdsetfsuffix(cmdssptr cmds,char *fname,int i);
```

scmdsetfsuffix sets the file suffix number for file name fname to equal the integer given in i. If the file name is not recognized, a 1 is returned to indicate an error; otherwise a 0 is returned. scmdsetfnames has to have been called first.

```
int scmdopenfiles(cmdssptr cmds,int vb);
```

scmdopenfiles opens any output files that are listed in the nfile and fname elements of the command superstructure. They should all be closed before this function is called. The total file name that is opened for each file is the string in the root element of cmds, concatenated with the string in the froot element of cmds, concatenated with the fname string for the file. If the name in fname is "stdout" or "stderr" then the file pointer is set to point to stdout or stderr, respectively. If vb is 0, any prior file is simply overwritten; otherwise, this routine looks for existing files and asks the user if any existing files should be overwritten. The function returns 0 for success and 1 for failure, where failure might arise from the user saying that a file should not be overwritten or from the inability to open a file for writing. If a file could not be opened, an error message is displayed to stderr. Upon failure, structures are not freed.

`FILE *scmdoverwrite(cmdssptr cmds,char *line2);`
`scmdoverwrite` reads the first word in `line2`, which is supposed to be a file name, looks for that name in the `fname` list of file names, closes the file, and reopens it. That way, the file is made empty for overwriting. The function returns the new file pointer (which is also stored in the `fptr` list of `cmds`), or `NULL` for failure.

`FILE *scmdincfile(cmdssptr cmds,char *line2);`
`scmdincfile` reads the first word in `line2`, which is supposed to be a file name, looks for that name in the fname list of file names, closes the file, increments the file name by one, and opens the new file. The function returns the new file pointer (which is also stored in the `fptr` list of `cmds`), or `NULL` for failure. This is useful for creating file stacks.

`FILE *scmdgetfptr(cmdssptr cmds,char *line2);`
`scmdgetfptr` is a utility routine for use by commands that save data to files. It reads the first word from `line2`, which is supposed to be a file name, and looks it up in the `fname` list in the command superstructure. If it was found, the corresponding file pointer is returned; otherwise `NULL` is returned. Also, if `line2` is sent in as `NULL` then a pointer to `stdout` is returned.

`void scmdoutput(cmdssptr cmds);`
`scmdoutput` displays the output files, the queue of commands, and the command timing parameters to `stdout`.

Internal routine

`void scmdcatfname(cmdssptr cmds,int fid,char *str);`
`scmdcatfname` concatenates all the portions of a file name together, for file number `fid`, into the string `str`, which should already be allocated to size `STRCHAR`. If the total file name is too long, it is truncated at size `STRCHAR`.

Possible improvements

The command string is fixed at 256 characters, which could be too short for some commands. In particular, a reasonable command might be “multicommand” whose arguments are a list of commands that should be run sequentially. This would allow a block structured command language.